

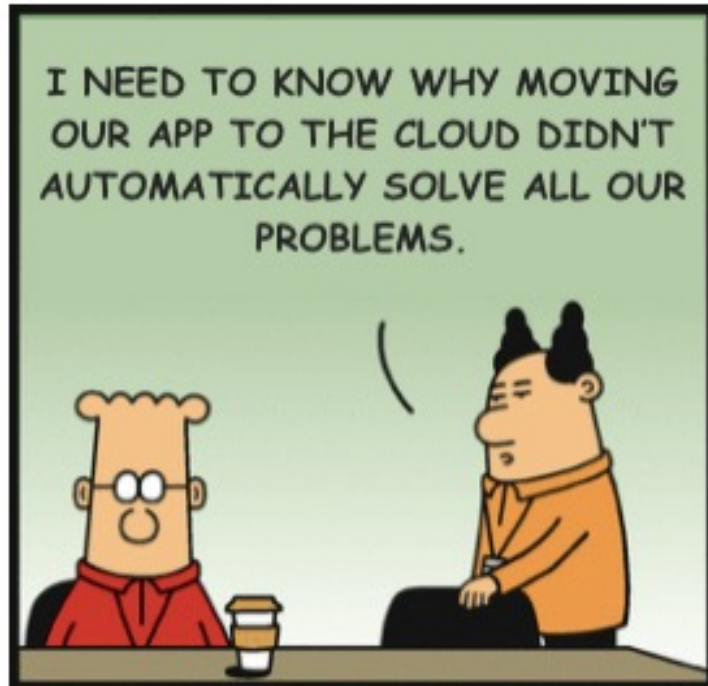
Cloud Native Architecture & Engineering

Introduction



Prof. Stefan Tai
Sebastian Werner, Elias Grünewald, Nicola Leschke, Maria Borges

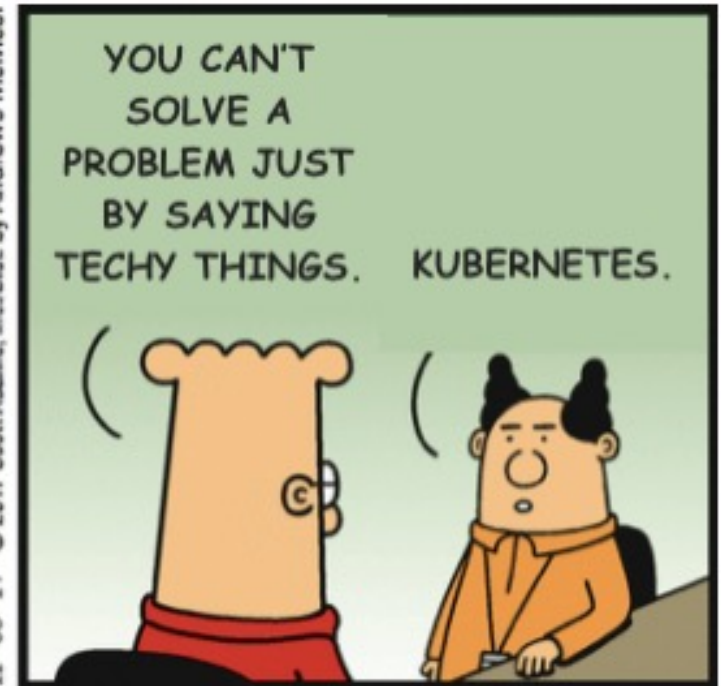
Information Systems Engineering
TU Berlin



Dilbert.com @ScottAdamsSays



11-06-17 © 2017 Scott Adams, Inc./Dist. by Andrews McMeel



Learning Objectives Today

Familiarize important terms (“keywords”) as CNAE concepts, which we will detail and study further in subsequent lectures

Be able to place these terms/concepts into a big CNAE picture, classify and categorize related terms in context

Remember system objectives associated with these terms and be able to ask appropriate questions with regards to meeting these objectives

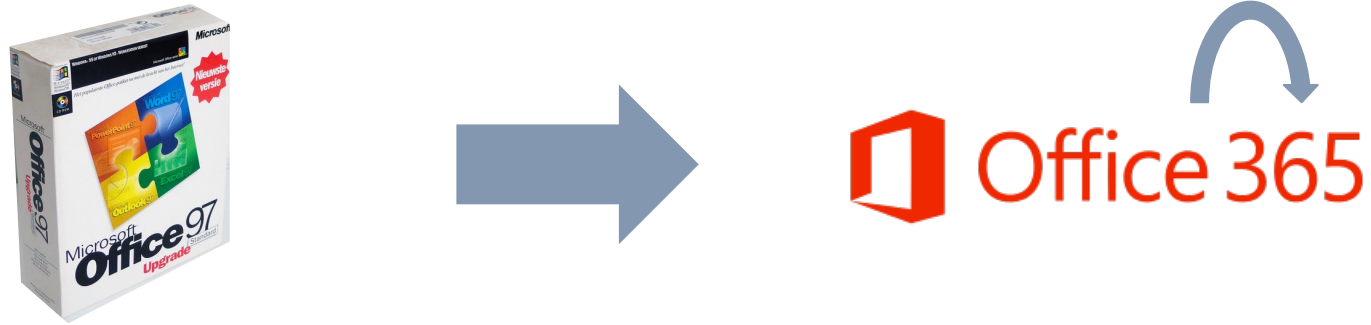
Understand the CNAE course roadmap and course elements and how these complement each other

Agenda



1. Intro / Brief Recap „Cloud Computing“
Distributed Systems and the Cloud
Software System Qualities / NFRs
2. Towards „Cloud-Native“
Microservices
Containers, Container Orchestration and Management
Serverless Computing and Function-as-a-Service
Cloud-Edge Continuum
3. Being Cloud-Native (and not „Cloud-Naive“)
Architecture and Engineering viewpoint: Migration, Refactoring, Continuous X
„Extreme“ Automation: IaC, Observability, and more

Changing Software Landscape

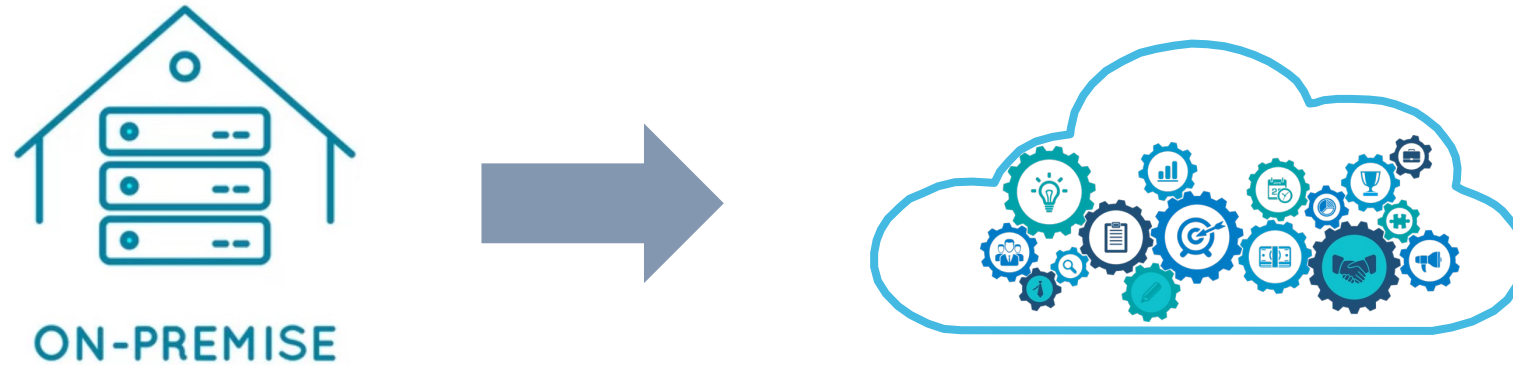


The internet has changed the software delivery landscape and driven the need for continuous change

Market opportunities appear or dissolve in months or weeks instead of years

Product differentiation, high quality or low cost are not enough. The competitive market demands speed and flexibility.

Increased Complexity of Software



Cloud computing facilitated the development of larger, more complex distributed systems

Larger systems are harder to manage and more prone to errors

Increased complexity drives the need to automate tasks and increase reliability

Recap: Cloud Computing

“Cloud computing is a model for enabling:

- ... convenient, on-demand network access

- ... to a shared pool of configurable computing resources
(e.g., networks, servers, storage, applications, and services)

- ... that can be rapidly provisioned and released with minimal management effort or service provider interaction“

- The NIST Definition [2010-2022]

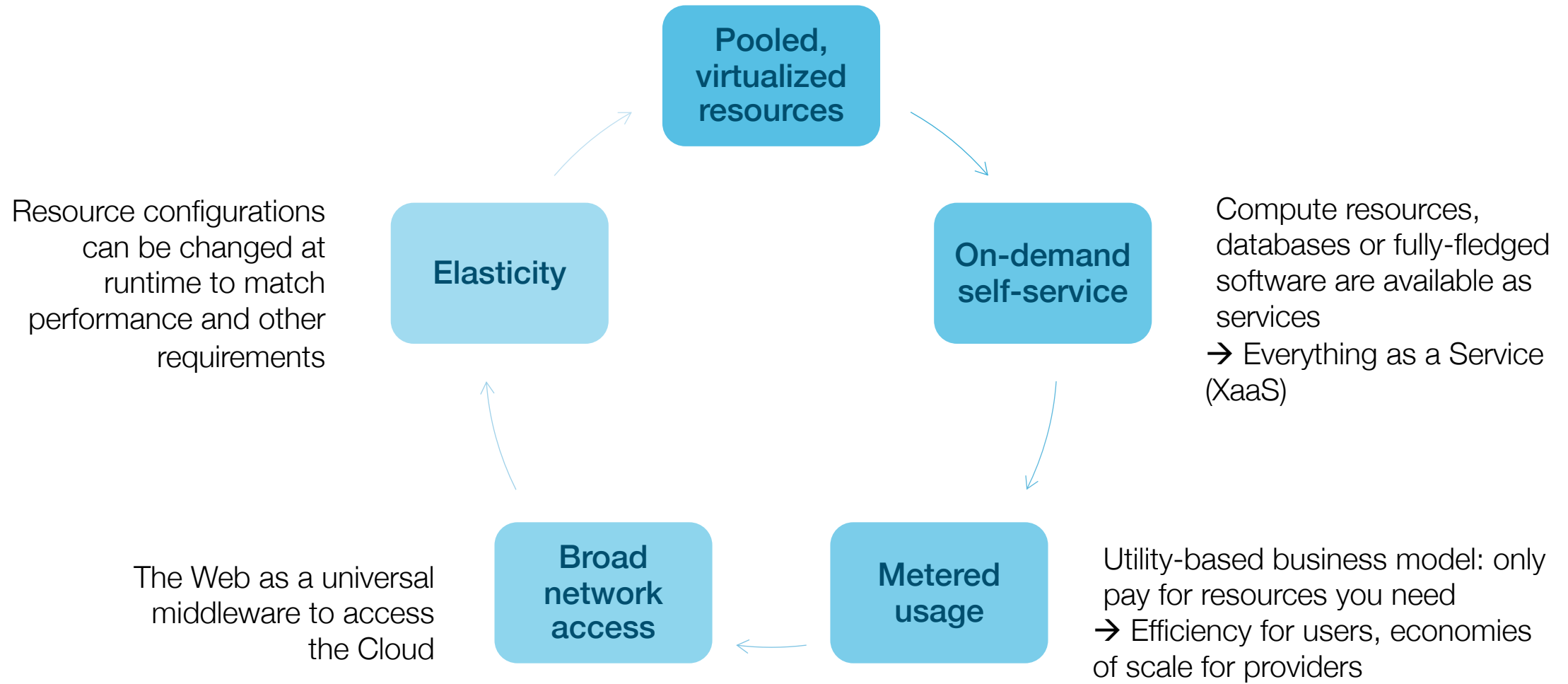
<https://csrc.nist.gov/projects/cloud-computing>

Quick Quiz:

What is the Cloud's business model and how does it translate into a cloud computing principle?

Recap: Cloud Principles

Efficient, pooled resources in large datacenters are dynamically allocated and aggregated or partitioned



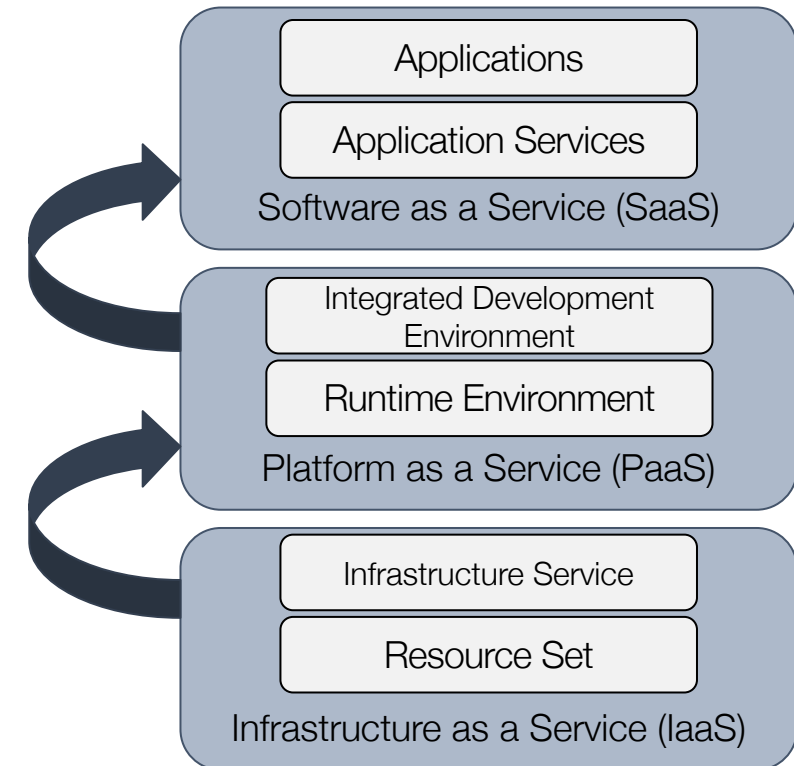
Cloud Computing Service Models

IaaS: Virtualized or physical resources and fundamental services (e.g. storage, computing, networking)

PaaS: Programming or execution environments, e.g. for the deployment of software artifacts written with certain frameworks or languages

SaaS: fully fledged applications, which replace a local installation for end-users (business software like CRM and office software, e.g. for spreadsheets)

No strict boundaries; services often offer features across adjacent layers



Cloud Computing Service Landscape



<https://aws.amazon.com/>

Google Cloud products

- Overview
- Featured products
- AI and Machine Learning
- API Management
- Compute
- Containers
- Data Analytics
- Databases
- Developer Tools
- Financial Services
- Healthcare and Life Sciences
- Hybrid and Multicloud
- Internet of Things (IoT)
- Management Tools
- Media and Gaming
- Migration
- Networking
- Operations
- Security and Identity
- Serverless Computing
- Storage
- More cloud offerings
- Product launch stages
- Take the next step

Google Cloud products

Browse over 100 products. New customers get \$300 in free credits to start running workloads and conduct an assessment.

[Get started for free](#)

Navigate to

[Product description cheat sheet](#)

[Pricing information](#)

[AWS and Azure comparisons](#)

[Industry solutions](#)

Featured products

Compute Engine

Virtual machines running in Google's data center.

Cloud Storage

Object storage that's secure, durable, and scalable.

Cloud SDK

Command-line tools and libraries for Google Cloud.

Cloud SQL

Relational database services for MySQL, PostgreSQL, and SQL Server.

Google Kubernetes Engine

Managed environment for running containerized apps.

BigQuery

Data warehouse for business agility and insights.

Cloud CDN

Content delivery network for delivering web and video.

Dataflow

Streaming analytics for stream and batch processing.

Operations

Monitoring, logging, and application performance suite.

Cloud Run

Fully managed environment for running containerized apps.

Anthos

Platform for modernizing existing apps and building new ones.

Cloud Functions

Event-driven compute platform for cloud services and apps.

<https://cloud.google.com/>

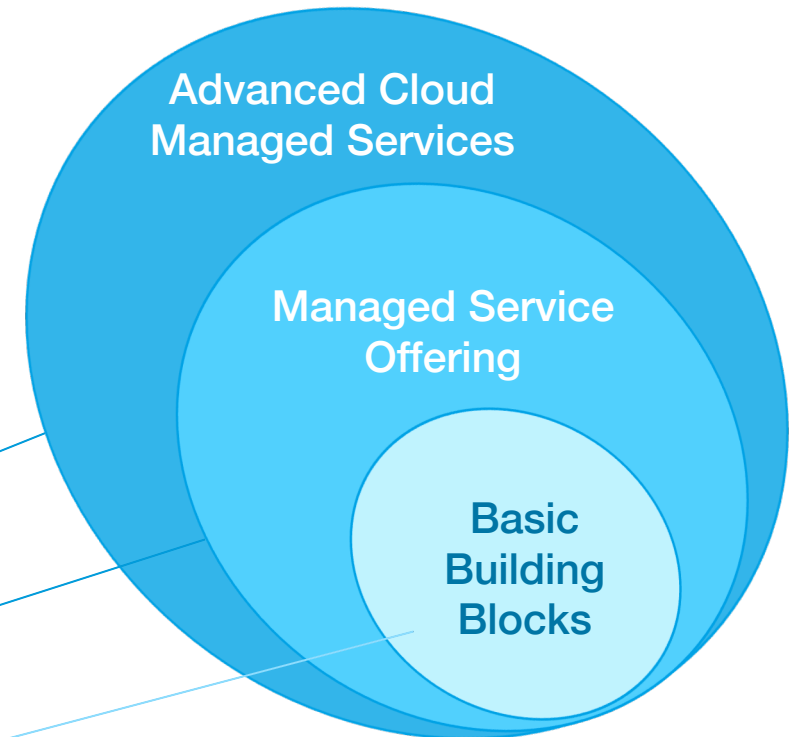
Cloud Service Offerings

- Each cloud vendor has its own service offering, the most mature have the richest set of features
- The incorporation of different services, from basic building blocks to the most advanced, cutting-edge technologies, can define how sophisticated a cloud native architecture is

Serverless offerings, ML and AI services, IoT services, ...

Managed databases, managed container services, managed messaging services, ...

VMs, simple storage, load balancers, ...



Agenda



1. Intro / Brief Recap „Cloud Computing“
Distributed Systems and the Cloud
Software System Qualities / NFRs
2. Towards „Cloud-Native“
Microservices
Containers, Container Orchestration and Management
Serverless Computing and Function-as-a-Service
Cloud-Edge Continuum
3. Being Cloud-Native (and not „Cloud-Naive“)
Architecture and Engineering viewpoint: Migration, Refactoring, Continuous X
„Extreme“ Automation: IaC, Observability, and more

Recap: Software System Qualities (aka Non-functional Requirements)

Nonfunctional Requirements		
Accessibility	Efficiency	Reliability
Availability	Extensibility	Responsiveness
Compatibility	Failure Transparency	Robustness
Certification	Fault Tolerance	Safety
Compliance	Flexibility	Scalability
Configurability	Interoperability	Securability
Dependability	Maintainability	Supportability
Deployability	Operability	Testability
Documentation	Performance	Traceability
Disaster Recovery	Recoverability	Usability

➔ How can we achieve these qualities? Each single one, multiples, all?

Quick Quiz:

Consider scalability.

How do we typically achieve scalability in cloud systems –
do we scale vertically or horizontally?

Recall: Scale out vs. scale up

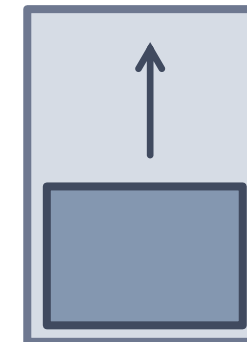
Horizontal Scalability

- Number of resources (e.g. servers) is increased if an application requires more processing power, storage etc.
- Not all architectures are designed to distribute their workload to multiple resources, so applications need to be adapted accordingly



Vertical Scalability

- Increasing performance of an application by increasing the capabilities of the IT resources on which it runs (without changing the number)
- This approach can also be used in the cloud, but it is limited by the capabilities of the provider



➔ Cloud applications should be *scaled out* rather than being scaled up

Scalability and Elasticity

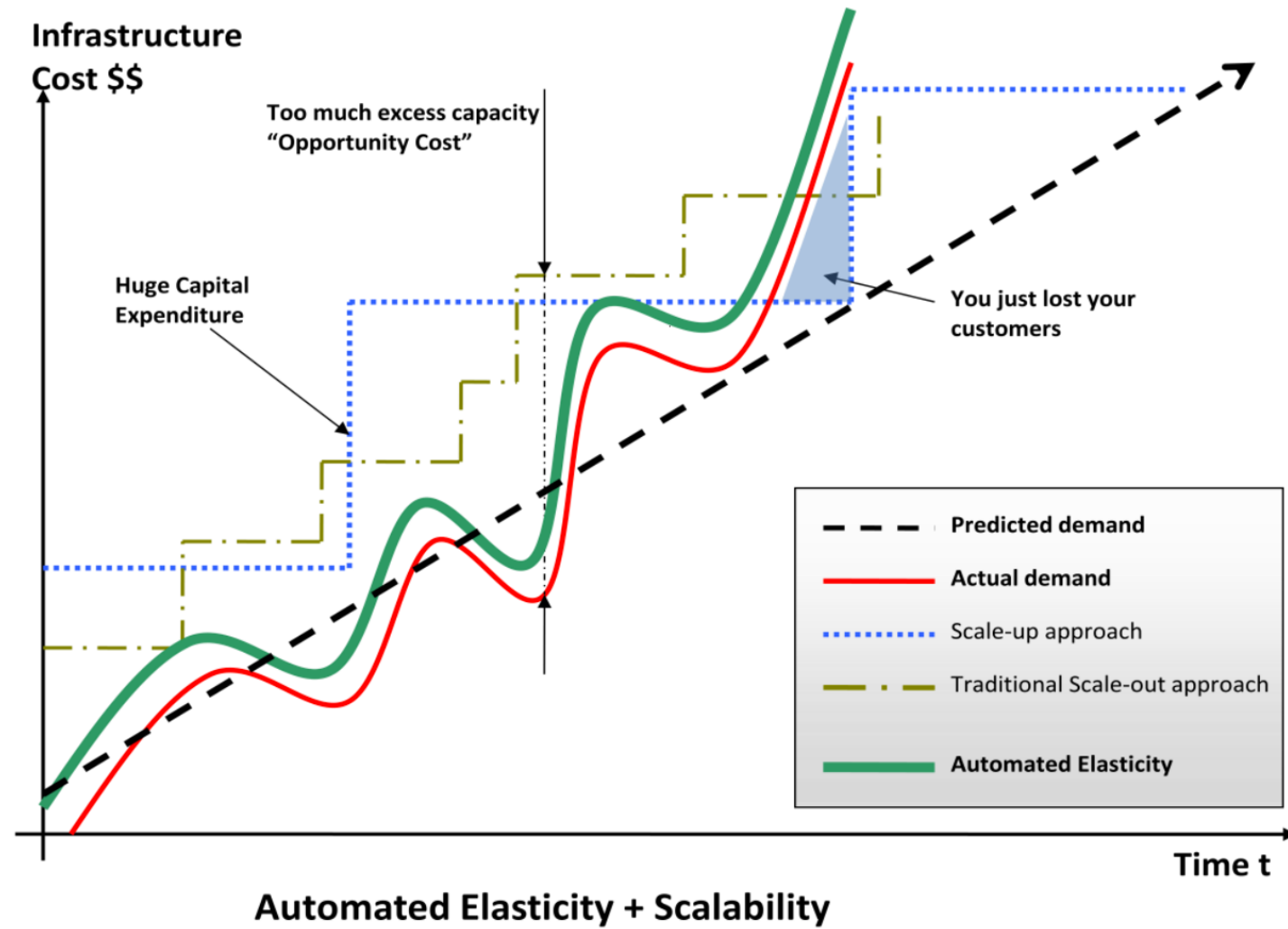


Scalability: increasing the amount of resources results in increased performance of an application

Elasticity: how well can an application handle changes to the amount of resources at runtime?

Elasticity implies Scalability

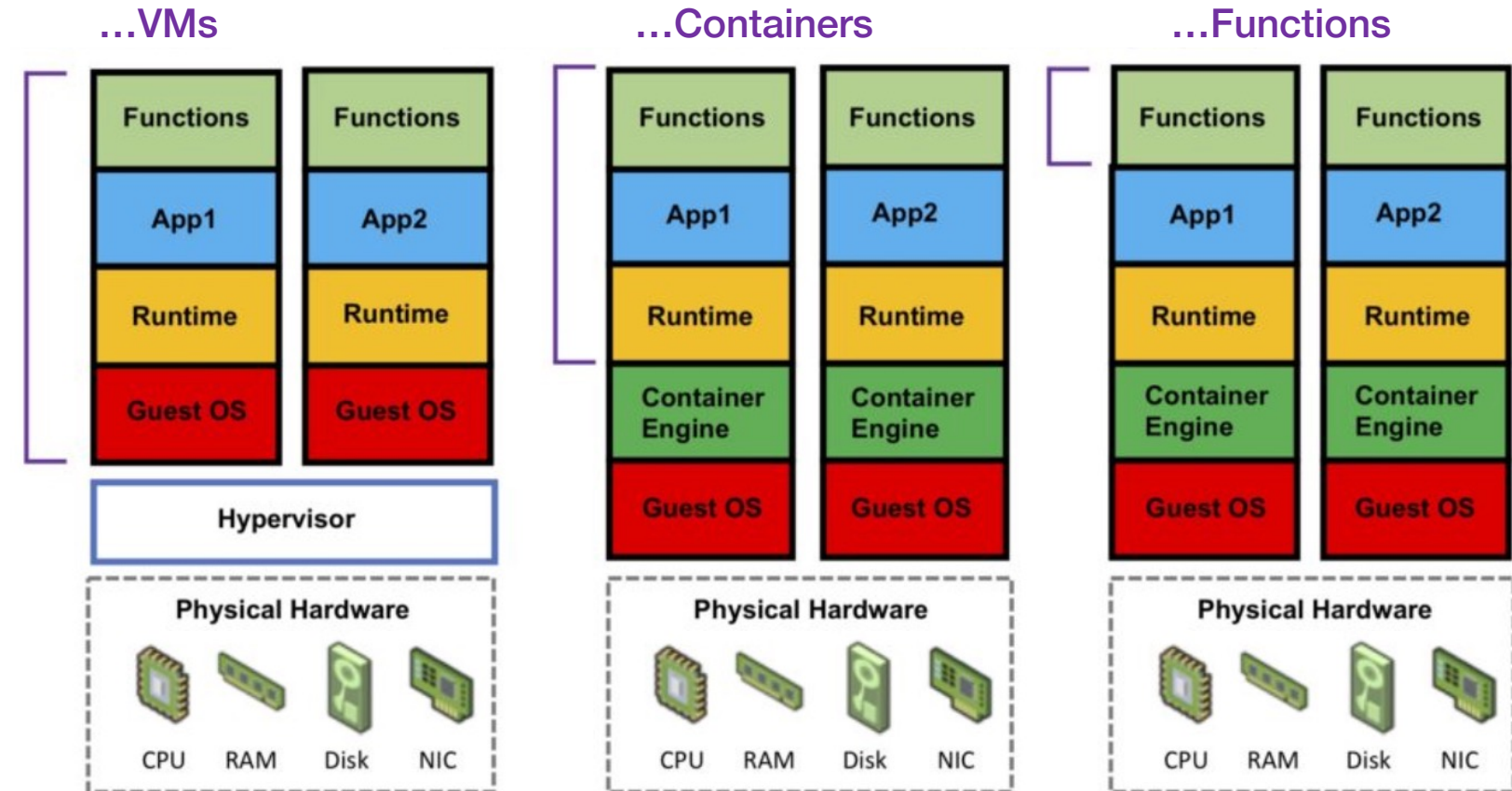
Elasticity



https://media.amazonwebservices.com/AWS_Cloud_Best_Practices.pdf

What are the units to scale, and who is responsible?

Consumer managed...



Recap: Software System Qualities (aka Non-functional Requirements)

Nonfunctional Requirements		
Accessibility	Efficiency	Reliability
Availability	Extensibility	Responsiveness
Compatibility	Failure Transparency	Robustness
Certification	Fault Tolerance	Safety
Compliance	Flexibility	Scalability
Configurability	Interoperability	Securability
Dependability	Maintainability	Supportability
Deployability	Operability	Testability
Documentation	Performance	Traceability
Disaster Recovery	Recoverability	Usability

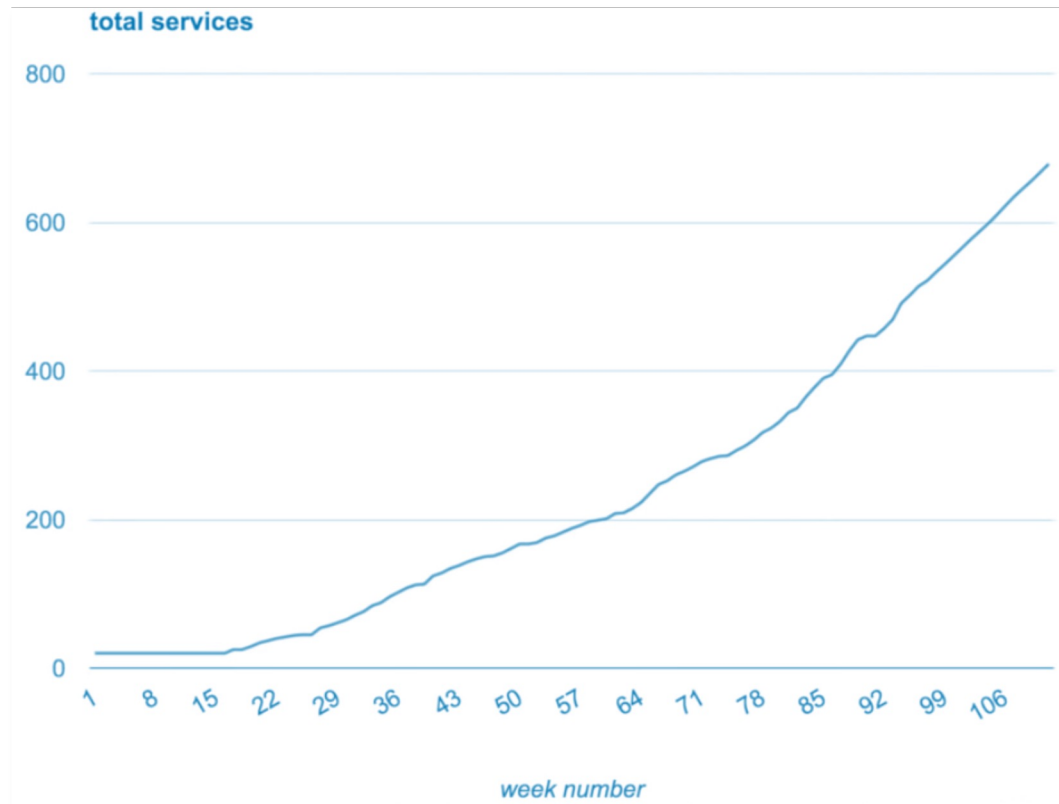
Agenda



1. Intro / Brief Recap „Cloud Computing“
Distributed Systems and the Cloud
Software System Qualities / NFRs
2. Towards „Cloud-Native“
Microservices
Containers, Container Orchestration and Management
Serverless Computing and Function-as-a-Service
Cloud-Edge Continuum
3. Being Cloud-Native (and not „Cloud-Naive“)
Architecture and Engineering viewpoint: Migration, Refactoring, Continuous X
„Extreme“ Automation: IaC, Observability, and more

Modern Enterprise Microservice Architectures

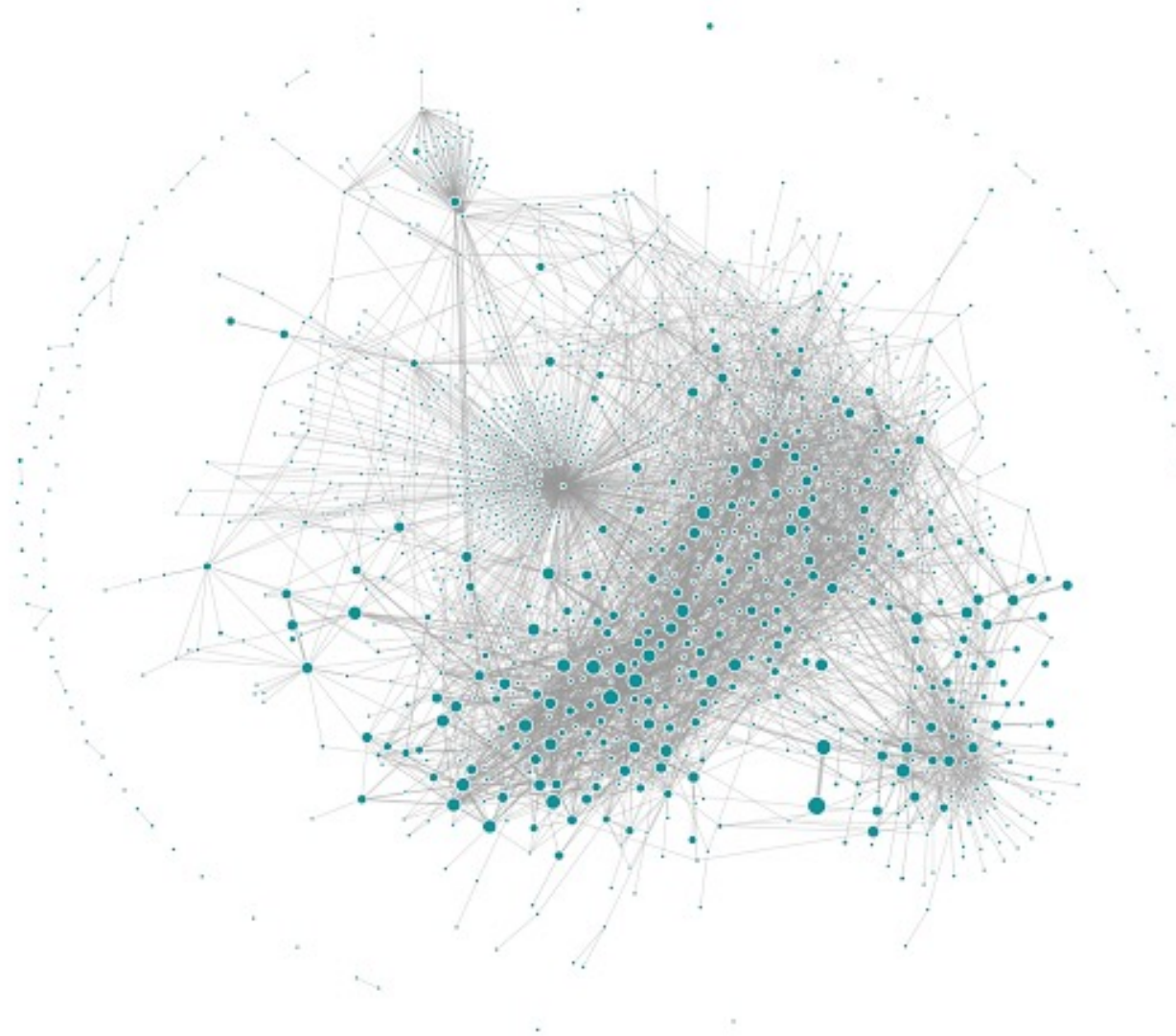
Uber 2016 to 2018



Uber 2019+

Services 4,000+	Mobile Apps 20+	Web Apps 500+	Dep Versions 10,000+
Languages 10+	Build Tools 10+	Repos 12,000+	Engineers 2,000+

Uber's microservice architecture ca. mid-2018 from Jaeger



Quick Quiz:

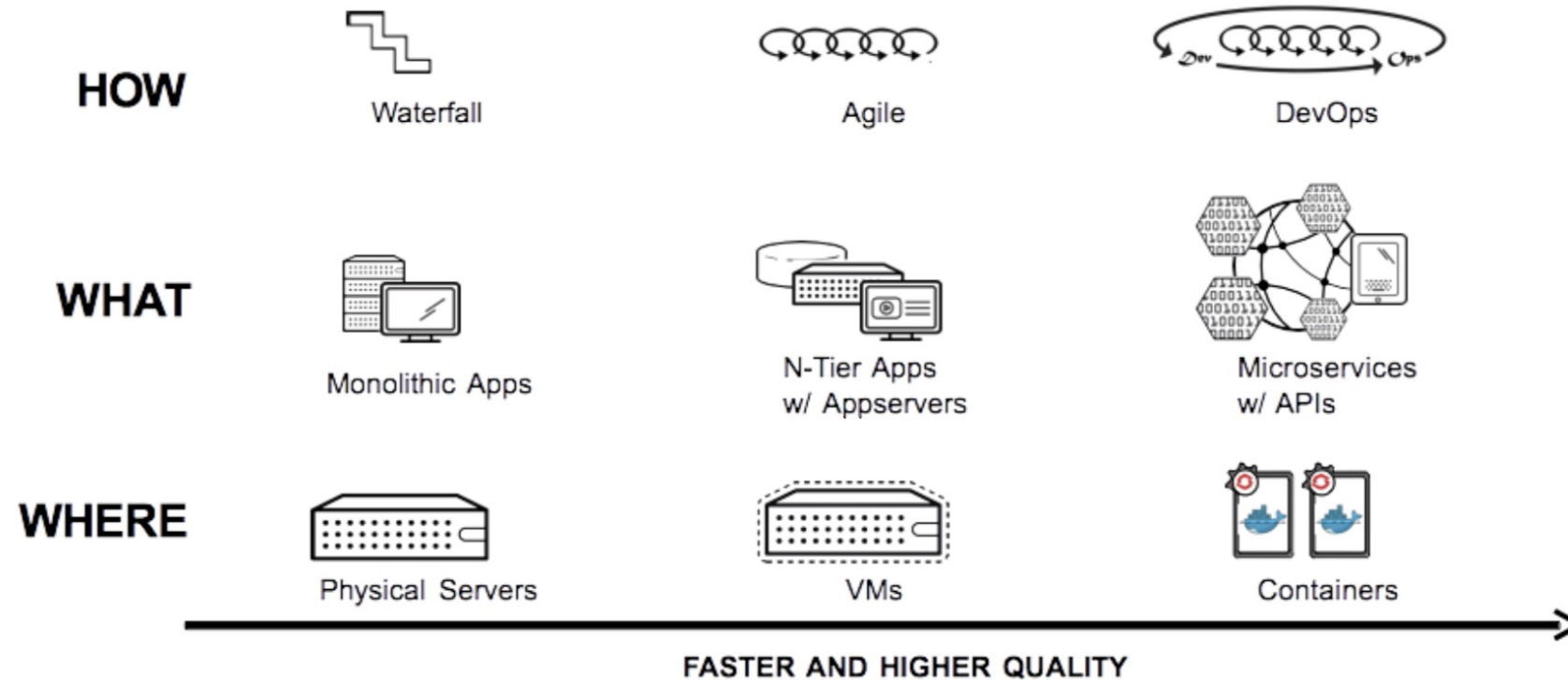
What is a Microservice, really?

“Microservices are an architectural and organizational approach to software development where software is composed of small independent services that communicate over well-defined APIs. These services are owned by small, self-contained teams.”

<https://aws.amazon.com/microservices/>

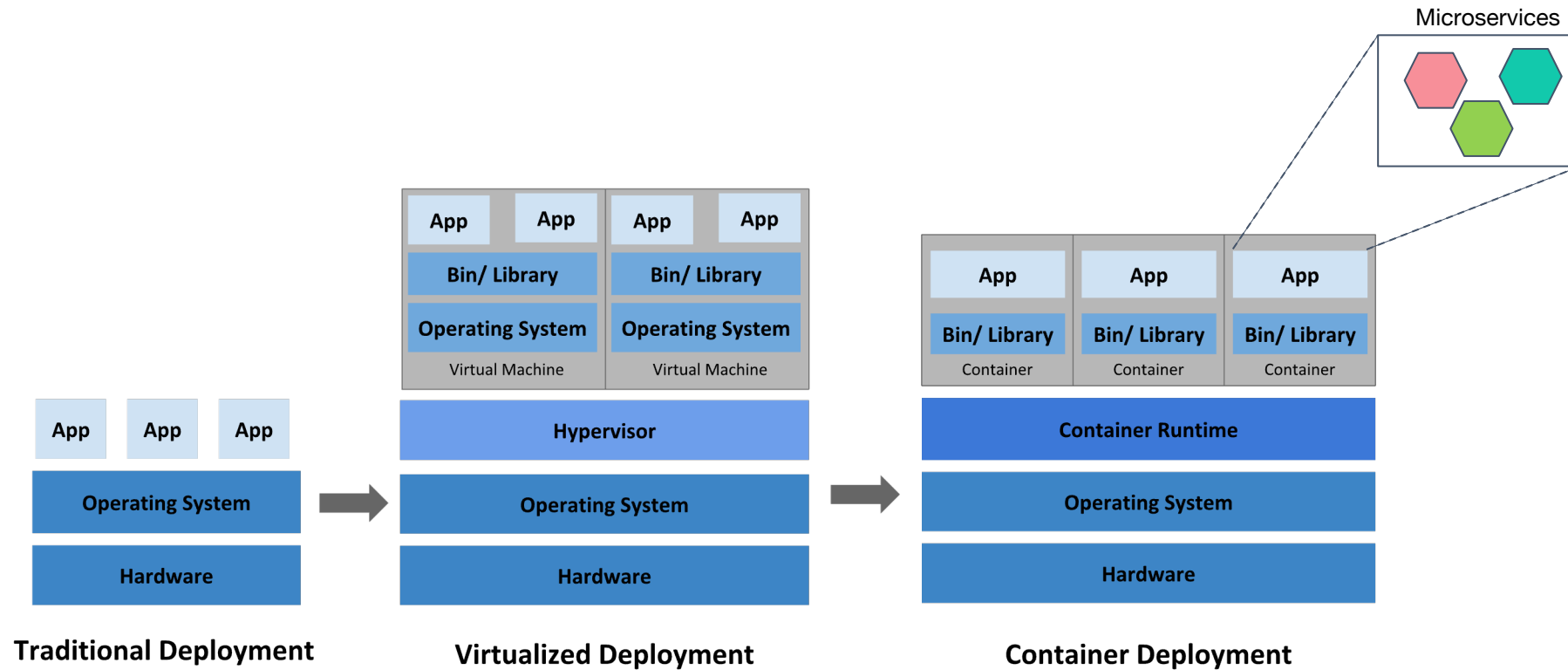
Find out more in the upcoming CNAE block on Microservices!

Evolution of Software Development



+ WHO

The Role of Containers



Source: <https://kubernetes.io/docs/concepts/overview/>

Quick Quiz:

Can you name some benefits of using Containers?

- Instant deployment
- Service discovery and load balancing
- Storage orchestration
- Automated rollouts and rollbacks
- Automatic bin packing
- Self-healing
- Secret and configuration management

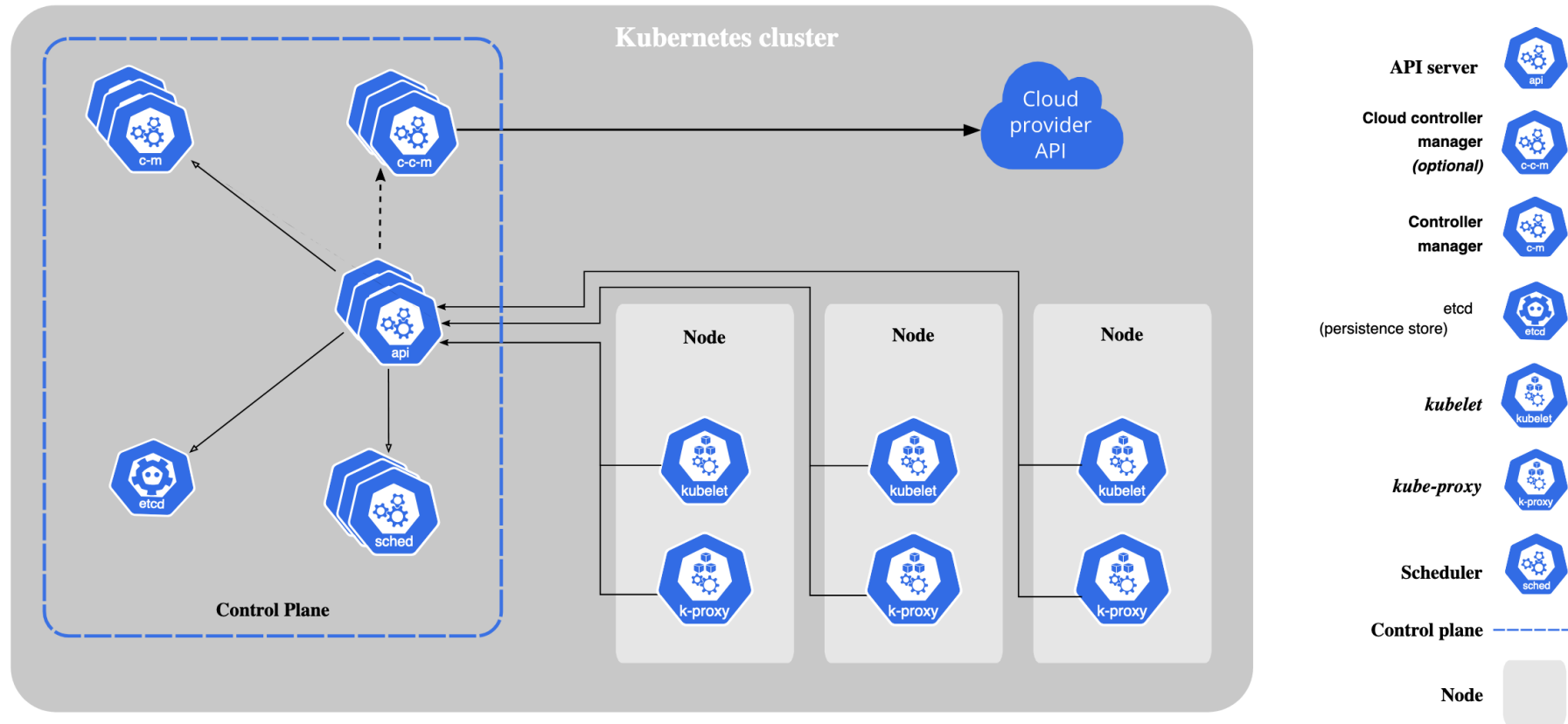
Find out more in the
upcoming CNAE block
on Containers!

Containers are the promised land!

Really?

Do not underestimate the complexity, efforts and skillsets needed to maintain the runtime environment – container runtime, data stores, messaging and related infrastructure, etc.

Container Orchestration



Components of a Kubernetes cluster, <https://kubernetes.io/docs/concepts/overview/components/>

Serverless – the “true” Cloud Native Paradigm?

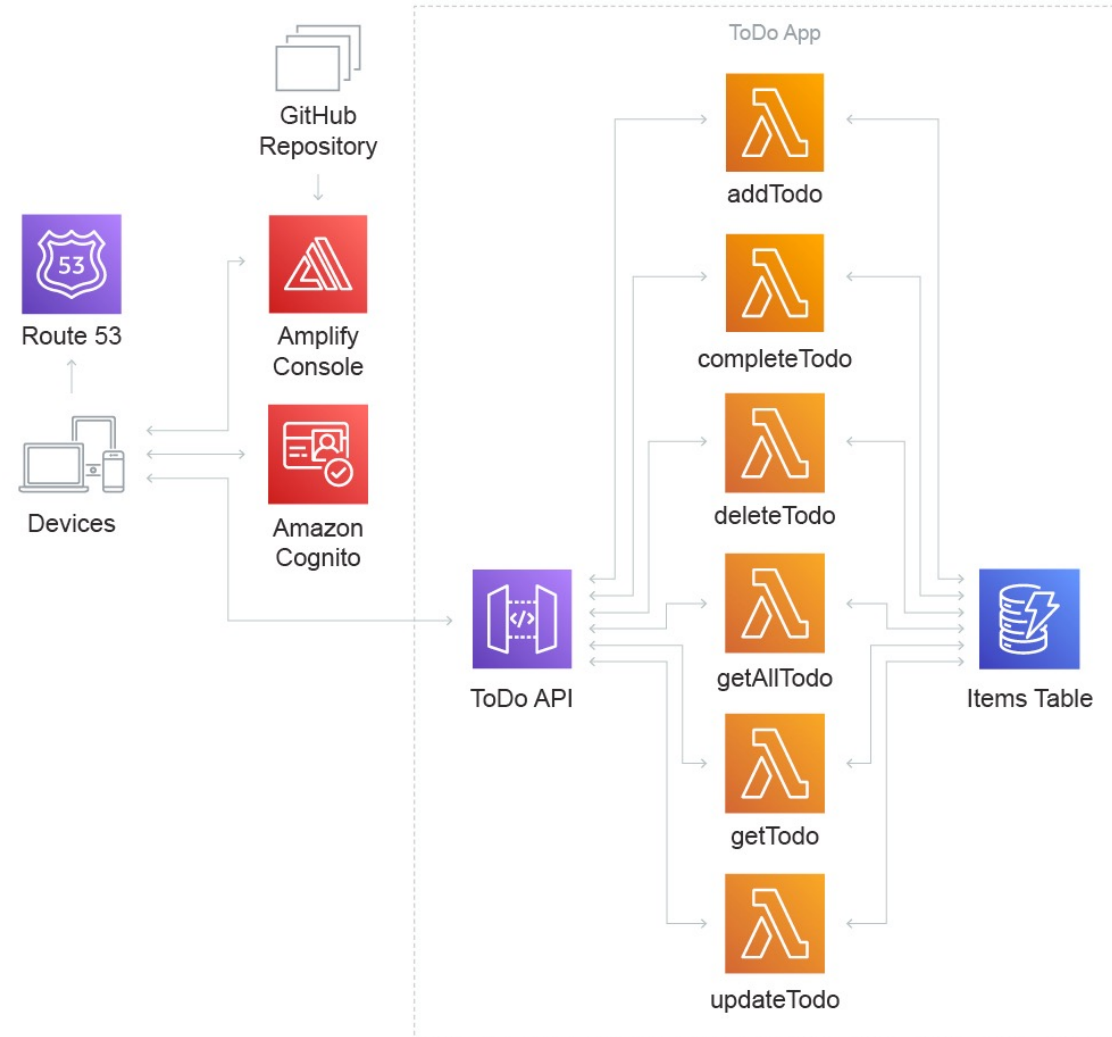
“Serverless” = The cloud consumer/application no longer worries about servers/infrastructure, but

- Let the cloud provider fully manage the underlying server infrastructure; have the consumer benefit from a fully managed cloud service
- Scale automatically (responsibility of the provider) and flexibly: scalability should follow actual invocation demand for business logic
- Correspondingly, have the consumer not pay for resources deployed, but for resources actually used

Find out more in the upcoming CNAE block on Serverless!

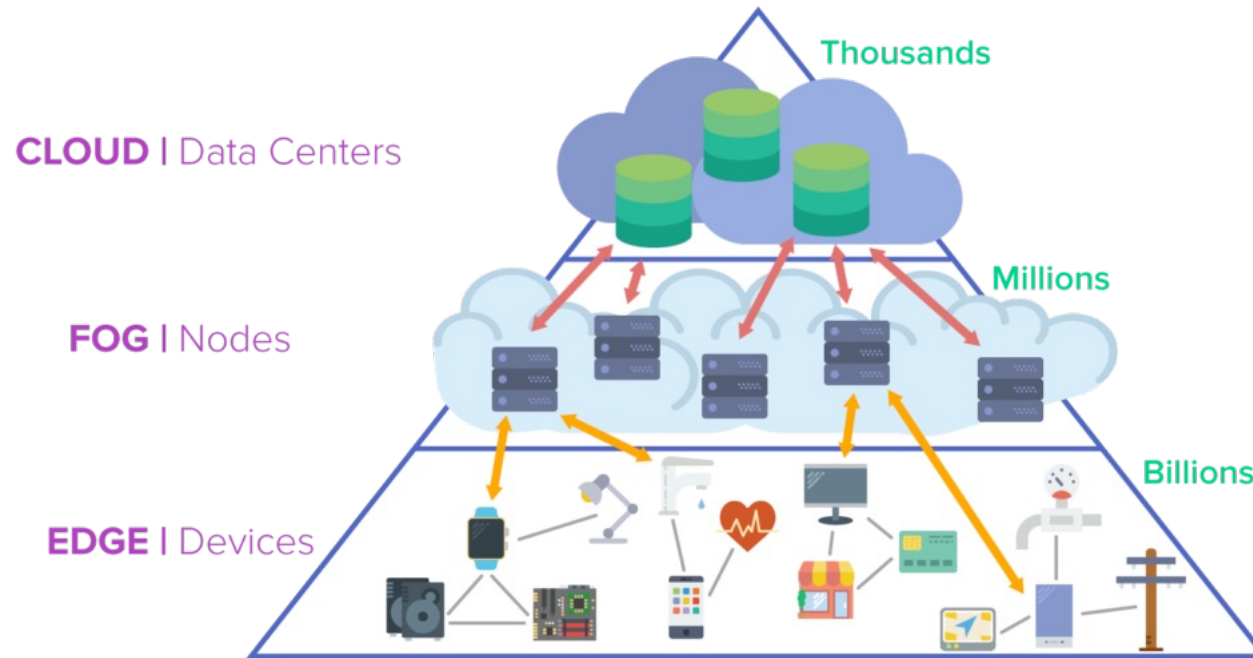
Build a simple “to-do list” Web App with AWS

- Event-driven backend
- API gateway (“reverse proxy”) for handling calls and results
- FaaS offering (here: AWS Lambda) to run code (“functions”) in response to events and to automatically manage the resources required by that code



Source: <https://aws.amazon.com/serverless/>

Where should code be executed, and data processed?



...towards a cloud-edge computing continuum

Agenda



1. Intro / Brief Recap „Cloud Computing“
Distributed Systems and the Cloud
Software System Qualities / NFRs
2. Towards „Cloud-Native“
Microservices
Containers, Container Orchestration and Management
Serverless Computing and Function-as-a-Service
Cloud-Edge Continuum
3. Being Cloud-Native (and not „Cloud-Naive“)
Architecture and Engineering viewpoint: Migration, Refactoring, Continuous X
„Extreme“ Automation: IaC, Observability, and more

Cloud Computing versus Cloud-Native

At its most basic form, cloud native means to embrace cloud computing services to design software solutions

However, that only covers part of what is required to become cloud native

The term “native“ implies development and operations tailored for (one or more) cloud platforms

Cloud Native Definition v1.0 by the CNCF

“Cloud native technologies empower organizations to build and run **scalable** applications in modern, **dynamic** environments such as public, private, and hybrid clouds.

Containers, service meshes, microservices, immutable infrastructure, and declarative APIs exemplify this approach.

“These techniques enable **loosely coupled** systems that are **resilient**, **manageable**, and **observable**.

Combined with **robust automation**, they allow engineers to make high-impact changes frequently and predictably with minimal toil.”



<https://www.cncf.io/>

Cloud Native



Build applications that take full advantage of the cloud

Tailored application design and automation play significant roles in the process

“Extreme automation” at scale to not only create instances or specific systems, but to also completely roll out an entire corporate landscape with little human interaction

Continuous software engineering methods are needed to achieve this

Being truly cloud-native implies – for some – embracing fully managed services, such as serverless offerings

Taking an Architecture and Engineering Perspective

How to divide (to-be) provided functionality among multiple logical application components (services, functions) that can be scaled out independently

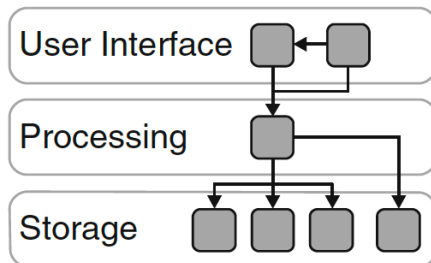
The independent components form an integrated distributed application

- ➔ How to decompose applications into different components?
- ➔ How to couple components into an integrated application?

Recall: Common Decomposition Approaches

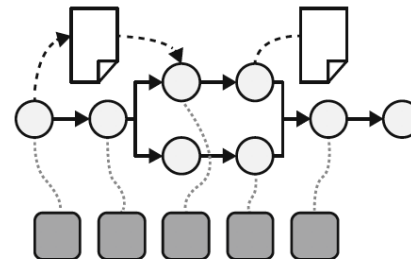
Layer-based

- Divides an application into separate logical layers
- Each layer is comprised of application components that provide a certain function
- Restricted access, avoids complicated interdependencies



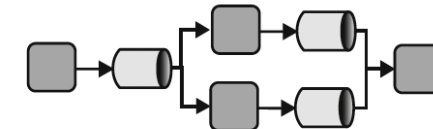
Process-based

- Focuses on business processes supported by the application
- Processes are comprised out of activities that are executed in a specific order
- Functionality accessed by the activities is decomposed into separate components



"Filters-and-pipes"-based

- Architecture for data-centric processing
- "Filters" provide a certain function performed on input data and produce an output
- Filters are interconnected with pipes ensuring that the output of one filter is fed to the next



Recall: Communication and Coupling

In order to work together, independent cloud components have to exchange information with each other

Different components may use different programming languages, data formats, and execution environments

Dependencies between components regarding their location, availability, and data format

Different approaches for communication between components can relax the dependency and create a more robust and flexible architecture

Tight versus loose coupling

Implications on Architecture

How to decompose applications into different components?

- Services are built around business capabilities and are independently deployable
- Decentralized data management

How to couple components into an integrated application?

- Communication through lightweight mechanisms
- Aim to be as decoupled and as cohesive as possible
- “Smart endpoints and dumb pipes”

→ Microservice Principles

Migrating to Cloud-Native

Migration Strategies:

- Lift and Shift
- Improve and Move
- Remove and replace

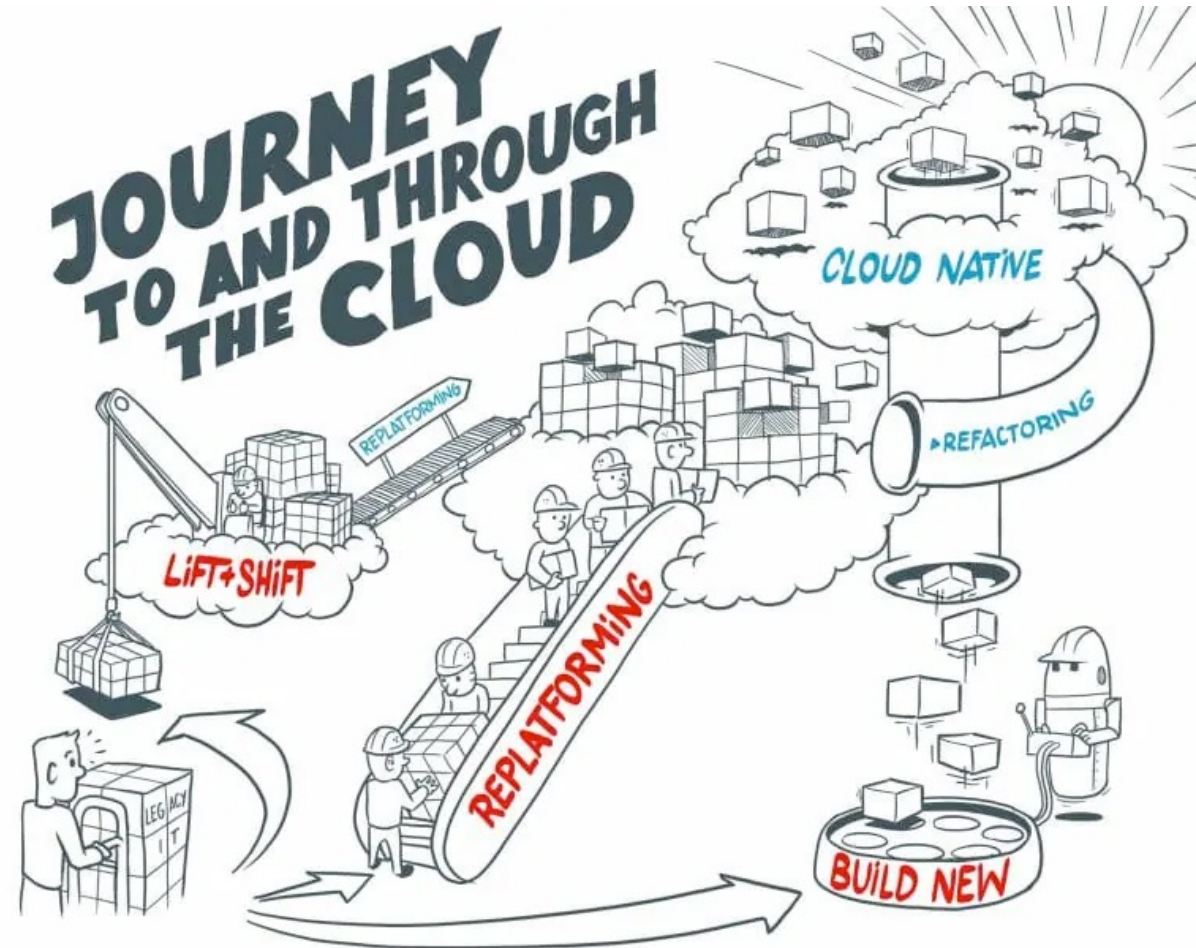


Figure source: claranet

(Code) Refactoring and Architecture Refactoring

(Code-) “Refactoring is a disciplined technique for restructuring an existing body of code, altering its internal structure without changing its external behavior.

Its heart is a series of small behavior preserving transformations. Each transformation (called a "refactoring") does little, but a sequence of these transformations can produce a significant restructuring. Since each refactoring is small, it's less likely to go wrong. The system is kept fully working after each refactoring, reducing the chances that a system can get seriously broken during the restructuring.”

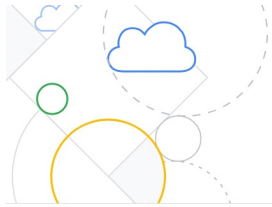
- Martin Fowler, refactoring.com

The main goal of architecture refactoring, however, is to increase the overall software system quality and surpass architecture and/or infrastructure limitations related to some system quality of concern.

(Some, like Fowler, may call this “restructuring”).

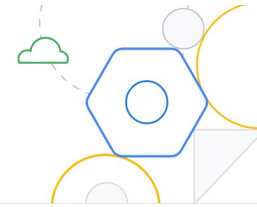
This year's CNAE practical assignment will deal with cloud native architecture refactoring

Best Practices [https://cloud.google.com/architecture]



Cloud Architecture Center

Discover reference architectures, diagrams, design patterns, guidance, and best practices for building or migrating your workloads on Google Cloud.



Design best-practice architectures on Google Cloud

The [Architecture Framework](#) helps you design a Google Cloud architecture that supports your business needs. Start with system design, or explore another category.

 [System design](#)

 [Reliability](#)

 [Operational excellence](#)

 [Cost optimization](#)

 [Security and privacy](#)

 [Performance optimization](#)

FILTER BY

Choose a topic

[Select all](#)

- ☐ Security and compliance **FEATURED**
- ☐ Big data and analytics **FEATURED**
- ☐ Artificial intelligence and machine learning (AI/ML) **FEATURED**

Filter results

Refactoring a monolith into microservices

This reference guide is the second in a four-part series about designing, building, and deploying microservices. This series describes the various elements of a microservices architecture. The serie...

Building production-ready data pipelines using Dataflow: Developing and testing data pipelines

This document explains best practices you should consider as you're developing and testing your...

Implementing canary deployments with Spinnaker and Istio

Spinnaker is an open source, continuous delivery system led by Netflix and Google. It's used to manage application deployment on various...

Quick Quiz:

Is this code or architecture to be refactored?

```
resource "google_compute_instance" "vm_instance" {  
  name = "terraform-instance"  
  machine_type = "f1-micro"  
  initial_node_count = "3"  
  boot_disk {  
    initialize_params {  
      image = "debian-cloud/debian-9"  
    }  
  }  
  network_interface {  
    network = google_compute_network.vpc_network.name  
    access_config {  
    }  
  }  
}
```


Infrastructure Automation and IaC

Deployment requires a configured environment for a software artifact

Infrastructure Automation deals with the automated provisioning and configuration of environments

Automated provisioning of infrastructure is enabled by **Infrastructure as Code**

Allows environments to be automatically created and configured, in a repeatable fashion

Requires **declarative models and API-driven infrastructure management**

Importance of Automation

Advantages of the cloud are emphasized by automating operational aspects of the application

Still,
Designing, deploying, and managing an application in the cloud is complicated. Modern enterprise microservice architectures are increasingly complex and of growing scale.

- This further complicates the task of managing services in the cloud

In the cloud, processes can become more streamlined and consistent by developing the environment and configuration process **with code**

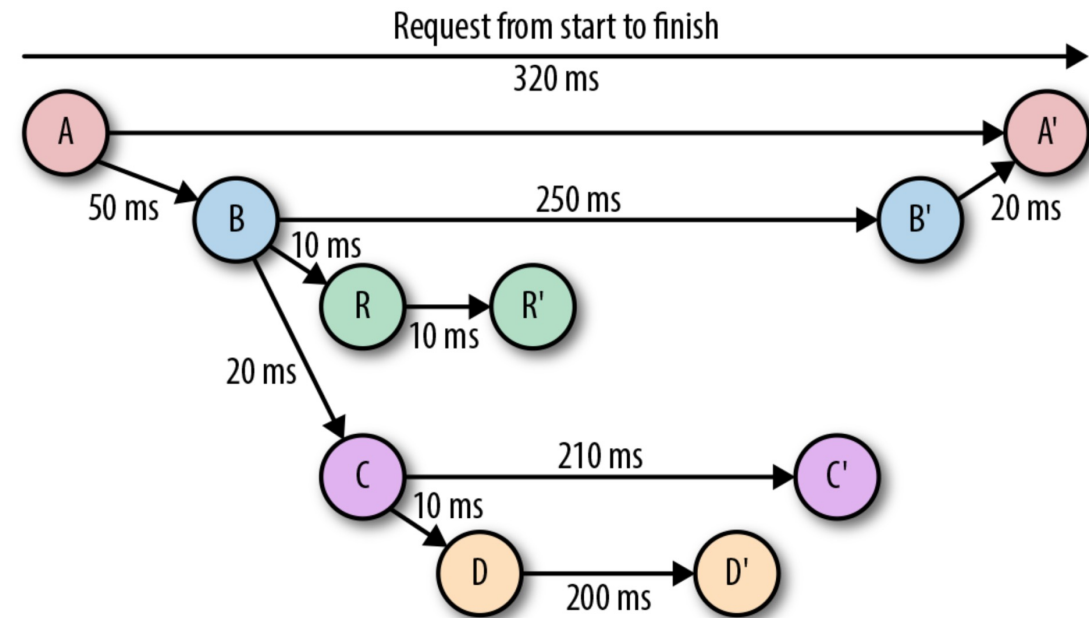
Quick Quiz:

How much of this is Dev, how much of this is Ops?

Observability: Proactive Prediction for a running Cloud System

Three pillars of observability: Metrics, Logs, and Traces

Example trace (consisting of spans):



Lifecycle of a request, represented as a directed acyclic graph
Figure source: C. Sridharan, Distributed Systems Observability, O'Reilly 2018

Site Reliability Engineering (SRE)

“Protect, provide for, and progress the software and systems behind services with an ever-watchful eye on their availability, latency, performance, and capacity.”

“SRE is what you get when you treat operations as if it’s a software problem.”

- Google, sre.google

The CNAE block on “Being Cloud-Native” will cover DevOps processes, architecture principles and best practices, aspects of automation, observability, and SRE.

Summary: Cloud Native – Value Proposition

Constrain costs

Increased business agility

Continuous X by

Experimentation → try out new services or build new features with low sunk costs in case of failure

Moving fast → rapid integration of new and managed services, without the operational overhead

CNAE Schedule (preliminary)

